Hangman

Résumé du Projet :

Les étudiants travaillent individuellement pour réaliser un jeu de pendu dans le terminal en utilisant le langage go.

Objectifs Pédagogiques :

- **Appliquer les bases de l'algorithmie :** Utiliser les concepts d'algorithmie (boucles, conditions, variables, fonctions) pour construire un jeu interactif et dynamique.
- **Voir ou revoir de nouvelles notions :** Maîtriser la syntaxe et les particularités du langage Go, notamment la gestion des erreurs, les slices, les maps et les packages.
- **Organiser et présenter un projet :** Structurer le projet de manière modulaire et professionnelle avec une hiérarchie de dossiers adéquate
- **Partager et garder un historique du code :** Utiliser un système de contrôle de version (Git) pour versionner le projet et partager le code de manière collaborative

---

Fonctionnalités Détaillées :

1. **Lecture du fichier de mots :**
    a. Le programme doit lire un fichier texte contenant une liste de mots (un mot par ligne), fourni en argument lors de l'exécution.
    b. Si le fichier est absent ou illisible, le programme doit afficher un message d'erreur clair et quitter proprement.

2. **Choix aléatoire d'un mot :**
    a. Le programme doit sélectionner aléatoirement un mot parmi ceux présents dans le fichier pour chaque nouvelle partie.

3. **Révélation de lettres aléatoires :**
    a. Au début de chaque partie, quelques lettres du mot sélectionné doivent être révélées aléatoirement pour aider le joueur à démarrer.
    b. Le nombre de lettres révélées est configurable (par exemple, 1 ou 2 lettres).

4. **Gestion du scénario de jeu :**
    a. **Victoire** : La partie se termine lorsque le joueur a deviné toutes les lettres du mot.
    b. **Défaite** : La partie se termine si le joueur atteint un nombre prédéfini d'erreurs (par exemple, 6 erreurs). Chaque mauvaise lettre soustrait un essai, et deviner un mot entier incorrect retire deux essais.

5. **Gestion des erreurs et réussites :**
    a. Le programme doit décrémenter le nombre d'essais restants à chaque tentative incorrecte.
    b. Les lettres correctes proposées par le joueur doivent être révélées dans le mot.
    c. Si le joueur propose un mot entier et que ce n'est pas le bon, deux essais sont retirés.

6. **Gestion des lettres déjà proposées :**
    a. Le programme doit mémoriser les lettres déjà soumises et empêcher qu'elles soient proposées à nouveau.
    b. Les lettres déjà tentées doivent être affichées au joueur pour éviter des répétitions.
7. **Passage du fichier de mots en paramètre :**
    a. Le fichier contenant les mots à deviner doit être passé comme argument lors du lancement du programme. Si le fichier n'est pas fourni, le programme doit informer l'utilisateur.
8. **Versionnage avec Git :**
    a. Le projet doit être versionné avec Git, permettant un suivi clair de l'évolution du code. Les étudiants doivent créer plusieurs commits montrant la progression des différentes étapes.
9. **Organisation du projet :**
    a. Utilisation d'un fichier "go.mod" pour gérer les dépendances du projet et organiser le code de manière structurée.

Étapes à suivre :

1. **Initialiser le projet :**
    - Créez un nouveau projet en initialisant un dépôt Git.
    - Créez un fichier "go.mod" pour la gestion des dépendances.

2. **Lecture et traitement du fichier de mots :**
    - Implémentez la fonction de lecture du fichier en paramètre.
    - Vérifiez la validité du fichier et gérez les erreurs potentielles.

3. **Choix aléatoire du mot et révélation de lettres :**
    - Sélectionnez un mot aléatoire parmi ceux du fichier.
    - Révélez quelques lettres au joueur de manière aléatoire au début de la partie.

4. **Implémentation du scénario de jeu :**
    - Gérer la victoire et la défaite en fonction des essais et des lettres découvertes.
    - Créer un compteur d'erreurs et décrémenter ce dernier à chaque lettre incorrecte.
    - Ajouter la possibilité pour le joueur de proposer un mot entier.

5. **Suivi des lettres proposées :**
    - Stockez les lettres déjà proposées par le joueur et affichez-les.
    - Bloquez la soumission de lettres déjà tentées.

6. **Gestion des arguments et lancement du programme :**
    - Ajoutez la gestion des arguments pour passer le fichier de mots en paramètre.
    - Testez le programme avec plusieurs fichiers de mots.

7. **Versionnage Git :**
    - Faites des commits réguliers à chaque étape de développement.
    - Utilisez les branches Git pour expérimenter ou tester des fonctionnalités.

8. **Finalisation et démonstration :**
    - Présentez une version finale du jeu avec une démonstration.
    - Assurez-vous que le projet est bien structuré avec un fichier "go.mod".

Livrables :

1. **Code source complet** du jeu incluant tous les fichiers .txt et les ressources.
2. **Fichier README** détaillant comment l'utilisateur doit lancer le jeu.
